

# Clase 3 — 07.11.25

#VSC



Profesora: Sara Gonzalo



Desarrollo de interfaces. JAVASCRIPT, JQUERY, REALIDAD VIRTUAL



Clase 3 — 07/11/2025



**Tema:** Final Accesibilidad web — Inicio Java Script

## 1 3

## Técnicas fundamentales HTML (WCAG)

Las WCAG 1.0 del W3C incluyen 14 pautas principales de accesibilidad. Para aplicarlas correctamente, se definieron una serie de **técnicas** que facilitan la accesibilidad, coherencia visual y compatibilidad con ayudas técnicas.

### 13.1 Reducir el mantenimiento y aumentar la coherencia

#### Pauta:

Crear un estilo de presentación coherente en todas las páginas.

#### Técnicas:

- Utilizar un número reducido de hojas de estilo.
- Usar hojas de estilo vinculadas en vez de estilos inline.
- Reutilizar nombres de clase ("class") para conceptos equivalentes en todas las hojas de estilo.

### 13.2 Permitir al usuario redefinir los estilos

#### Pauta:

Evitar características desaconsejadas por el W3C.

#### Técnicas:

- En CSS2, cualquier regla marcada como `!important` en la hoja del usuario tiene prioridad sobre la del autor.

### 13.3 Unidades de medida

#### Pautas:

- Definir tamaños con unidades relativas ( `em` , `rem` , `%` ) en lugar de absolutas.
- Usar medidas absolutas solo cuando el medio físico lo requiera (p. ej., imágenes bitmap).

#### Técnicas:

- Definir tipografía con `em` .
- Usar porcentajes para tamaños y proporciones.
- Reservar unidades absolutas solo para casos donde la salida física esté completamente definida.

### 13.4 Contenidos generados

#### Pautas:

- Utilizar marcadores apropiados en lugar de imágenes.
- El documento debe ser legible incluso sin CSS.

#### Técnicas:

- Incluir equivalentes textuales para todo contenido generado por CSS ( `background-image` , `content` , `list-style` ).
  - Garantizar que todo el contenido importante esté en el HTML, no solo en la hoja de estilo.
  - El texto generado con CSS **no aparece en el DOM**, por lo que no es accesible para lectores de pantalla.
- 

## 13.5 Tipos de letra

#### Pautas:

Evitar elementos HTML obsoletos o desaconsejados.

#### Técnicas:

- Definir siempre un tipo de letra genérico por defecto.
  - Usar propiedades CSS modernas:
    - `font-family` , `font-size` , `font-weight` , `font-stretch` , `font-style` , `font-variant` , etc.
  - Evitar `<font>` , `<basefont>` , `face` y `size` .
  - `<big>` y `<small>` se permiten si es estrictamente necesario.
- 

## 13.6 Efectos de estilo del texto

#### Pautas:

Evitar efectos no soportados por todos los dispositivos.

#### Técnicas:

- Reducir transformaciones complejas ( `text-transform` ).
  - Usar con moderación:
    - `text-shadow`
    - `text-decoration`
  - Evitar efectos visuales que dependan de destellos o parpadeos.
- 

## 13.7 Texto en vez de imágenes

#### Pautas:

Priorizar el uso de CSS frente a imágenes para maquetación.

#### Técnicas:

- Emplear texto estilizado con CSS en lugar de imágenes.
  - Beneficios:
    - Mejora accesibilidad (lectores de pantalla, braille, dispositivos de voz).
    - Permite al usuario personalizar tamaños, colores y contraste.
    - Reduce carga innecesaria y mejora usabilidad.
-

## 13.8 Formateo y posición del texto

### Pautas:

Usar hojas de estilo para maquetación y presentación.

### Técnicas:

- Sangría: `text-indent` (no usar `<blockquote>` para simular sangría).
  - Espaciado: `letter-spacing`, `word-spacing`.
  - Control del espacio en blanco: `white-space`.
  - Dirección del texto: `direction`, `unicode-bidi`.
  - Pseudoelementos para destacarlo:
    - `::first-letter`
    - `::first-line`
- 

## 13.9 Colores

### Pautas:

- Asegurar contraste suficiente entre texto y fondo.
- Garantizar accesibilidad para usuarios con deficiencia de percepción de color.
- No basar información únicamente en el color.

### Técnicas:

- Usar códigos de color numéricos (hex o rgb).
  - Propiedades CSS:
    - `color`
    - `background-color`
    - `border-color`, `outline-color`
  - Gestionar colores de enlaces con: `:link`, `:visited`, `:active`
- 

## 13.10 Pistas de contexto en listas HTML

### Pautas:

- Marcar correctamente listas e ítems.
  - Usar UL/OL para aportar información semántica y claridad estructural.
- 

## 13.11 Maquetación, ubicación, capas y alineación

### Pautas:

- Utilizar CSS para maquetar, nunca tablas.
- Mantener separación entre estructura (HTML) y presentación (CSS).

### Técnicas:

- Alineación: `text-align`.
- Espaciado externo: `margin`, `margin-top`, `margin-bottom`, etc.
- Posicionamiento mediante:
  - `float`

- `position` (`absolute`, `relative`, `fixed`)
  - `top`, `right`, `bottom`, `left`
  - Estas propiedades permiten crear barras laterales, encabezados, pies de página y contenidos en capas sin romper semántica.
- 

## 13.12 Líneas y bordes

### Pautas:

- El documento debe poder leerse incluso sin hoja de estilo.
- Líneas y bordes pueden ayudar a la separación visual, pero no deben ser la única fuente de información (los usuarios que acceden sin CSS no los percibirán igual).

### Técnicas:

Utilizar propiedades CSS para definir bordes:

- `border`, `border-width`, `border-style`, `border-color`.
  - En tablas: `border-spacing`, `border-collapse`.
  - Para contornos dinámicos: `outline`, `outline-color`, `outline-style`, `outline-width`.
- 

## 13.13 Crear movimiento con hojas de estilo y scripts

### Pautas:

Hasta que los navegadores permitan detener cualquier animación, se recomienda **evitar movimientos excesivos**.

### Técnicas:

- Mostrar/ocultar contenidos.
  - Cambiar la presentación mediante movimiento suave o variación de colores (evitando efectos agresivos).
- 

## 13.14 Hojas de estilo en cascada auditivas

### Pautas:

Proporcionar la información de forma que se adapte a las preferencias del usuario (idioma, tipo de contenido, lectura hablada).

### Propiedades relevantes de CSS2 para audio:

- `volume` : controla el volumen del texto hablado.
  - `speak` : define si se pronuncia, y cómo (letra por letra, palabra por palabra).
  - `pause`, `pause-before`, `pause-after` : controlan pausas antes y después del contenido.
  - `cue`, `cue-before`, `cue-after` : permiten reproducir sonidos para orientar al usuario (equivalente auditivo a un icono visual).
- 

## Niveles de adecuación (Conformidad WCAG)

Las WCAG dividen la accesibilidad en **tres niveles**, que indican cuántas barreras elimina una web y qué grado de inclusión ofrece. Cada nivel corresponde al cumplimiento de un conjunto de criterios: prioridad 1 (A), prioridad 2 (AA) y prioridad 3 (AAA).

---

## ✓ Nivel A (Prioridad 1) — Accesibilidad mínima imprescindible

Es el **mínimo absoluto** que debe cumplir una web para que una persona con discapacidad pueda usarla sin quedar totalmente bloqueada.

**Si una página no cumple Nivel A, una parte del público directamente no podrá usarla.**

### Ejemplos de fallos típicos de Nivel A:

- Imágenes sin texto alternativo.
- Formularios sin etiquetas asociadas a sus campos.
- Navegación que depende solo del ratón (sin teclado no funciona).
- Contrastes extremadamente bajos.
- Contenido que parpadea rápidamente (riesgo de epilepsia).

### Objetivo del nivel A:

Evitar barreras críticas que impiden el acceso básico a la información.

---

## ✓✓ Nivel AA (Prioridades 1 y 2) — Accesibilidad recomendada

Es el **estándar profesional**.

Lo exigido por **las administraciones públicas, universidades, hospitales, bancos** y empresas medianas/grandes.

El Nivel AA corrige la mayoría de los problemas comunes de accesibilidad y asegura que prácticamente cualquier usuario pueda usar la web con **comodidad**.

### Ejemplos de requisitos AA:

- Contraste suficiente entre texto y fondo (no solo “visible”, sino dentro de los ratios recomendados).
- Textos reescalables sin perder estructura.
- Navegación consistente en todas las páginas.
- Estados visibles en los elementos interactivos (focus, hover).
- Evitar contenido dependiente exclusivamente del color.
- Evitar instrucciones vagas (“haz clic en el botón verde”).

### Objetivo del nivel AA:

Garantizar que la web es funcional, usable y coherente para la mayoría de usuarios con necesidades diversas.

---

## ✓✓✓ Nivel AAA (Prioridades 1, 2 y 3) — Accesibilidad máxima

Es el nivel **más estricto**.

Se aplica solo en sitios donde la accesibilidad es prioritaria:

- Salud
- Educación
- Inclusión social
- Administraciones con servicios críticos

El Nivel AAA no es “más bonito”, sino **más inclusivo**, especialmente con usuarios ciegos, sordos, con discapacidades cognitivas o motrices severas.

## Ejemplos de requisitos AAA:

- Subtítulos en directo para contenido multimedia.
- Traducción a lengua de signos.
- Contraste aún más estricto que el AA.
- Lectura fácil (“lectura plain language”).
- Navegación que evita cualquier carga cognitiva excesiva.
- Descripciones extendidas para imágenes complejas.

## Objetivo del nivel AAA:

Garantizar una experiencia optimizada incluso para usuarios con necesidades altamente específicas.

---

## Notas clave que debes dominar

### 1. El nivel mínimo exigido por ley: AA

Normalmente, las normativas obligan a cumplir AA (como la ley española **UNE-EN 301549**).

### 2. El W3C no certifica webs

No hay un “sello oficial”.

El cumplimiento se declara mediante:

- validaciones automáticas
- auditorías de accesibilidad
- auto-declaraciones del proveedor

### 3. Cada nivel tiene su logotipo oficial

El autor puede incluir el logotipo correspondiente **solo si realmente cumple los criterios**.

### 4. No todo contenido puede llegar a AAA

A veces es *técnicamente imposible* (p. ej., streaming en directo con subtítulo 100% perfecto).

---

## Herramientas de análisis de accesibilidad

Estas herramientas permiten validar automáticamente si una web cumple las WCAG.

#### • TAW (España)

Valida según WCAG 2.0. Permite elegir nivel A, AA, AAA.

Soporta HTML, CSS y JS.

| <https://www.tawdis.net/?lang=es>

#### • Observatorio de Accesibilidad Web de Ecuador

Valida por URL y permite elegir resolución para diseño responsive.

Permite niveles A, AA, AAA.

| <https://www.consejodiscapacidades.gob.ec/observatorio-de-accesibilidad-web/>

#### • Tenon

Extensión para Firefox, Chrome y Opera.

Analiza directamente la página activa.

## • Google Accessibility Developer Tools

Extensión de Chrome que evalúa automáticamente accesibilidad del sitio que estás visualizando.



axe DevTools - Web Accessibility Testing

[www.deque.com/](https://www.deque.com/) Featured

3.9 ★

Accessibility Checker for Developers, Testers, and Designers in Chrome

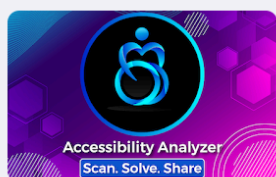


ARC Toolkit

[tpgi.com](https://tpgi.com) Featured

3.5 ★

Accessibility testing tool from TPGi

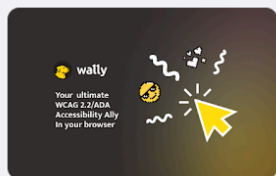


Accessibility Analyzer

[www.a11yanalyzer.com](https://www.a11yanalyzer.com)

5.0 ★

Perform Accessibility audits on webpages based on WCAG 2.2 and download free excel sheet reports



WAX: Wally Accessibility ADA/WCAG Compliance Checker and Fixer

[wallyax.com](https://wallyax.com)

5.0 ★

Wally AX's ADA/WCAG Compliance Checker and Fixer



## Cierre de la unidad de Accesibilidad

Sara comenta que esta unidad termina aquí.

Lo fundamental que debemos recordar:



### Los niveles de conformidad WCAG:

1. A → mínimo indispensable
2. AA → estándar recomendado
3. AAA → accesibilidad máxima

Son la base para cualquier auditoría o verificación seria.



## A partir de aquí: Comienza JavaScript

La profesora explica que el objetivo de las siguientes clases es:

✨ **Diseñar un sitio web completo utilizando solamente nodos de JavaScript, sin escribir HTML5 manualmente.**

Es decir:

- Crear estructuras del DOM desde JS.
- Insertar elementos dinámicamente ( `createElement` , `appendChild` , `querySelector` , etc.).
- Controlar estilos y eventos desde JS.

- Generar la página entera mediante scripts.

Sara dice que la meta es que el alumno comprenda profundamente **la relación entre JS, el DOM y la estructura de la web**, sin apoyarse en plantillas HTML.

---

## JAVASCRIPT – CLASE 3

### 1 ¿Qué es JavaScript?

JavaScript **no es un lenguaje independiente**:

funciona *dentro* de documentos HTML, incrustado entre etiquetas de marcado como:

```
<script></script>
```

Es un lenguaje orientado a trabajar con:

- **Objetos** (entidades predefinidas o creadas por el usuario)
  - **Eventos** (acciones del usuario o del sistema: `click`, `mouseover`, etc.)
  - **Acciones** que se desencadenan según la interacción
- 

### 2 Características principales

- **Lenguaje de marcado dentro de HTML** usando `<script>`.
  - **Case Sensitive**
    - `Hola` y `hola` no son lo mismo.
  - **Tipado dinámico**
    - no se especifica el tipo de dato.
    - JS realiza **conversión implícita** al asignar valores.
  - **Basado en objetos.**
  - **Existe un ecosistema enorme de frameworks:**
    - Angular
    - React
    - Vue
    - Node.js (lado servidor)
- 

### 3 Usos de JavaScript

- Crear sitios dinámicos (interacciones, animaciones, efectos).
  - Validar datos (formularios, expresiones regulares).
  - Trabajar con archivos externos (XML, JSON).
  - Modificar totalmente el CSS desde JS.
  - Crear peticiones **asíncronas** (AJAX).
  - Controlar eventos (`click`, teclado, ratón, `scroll`).
  - Construir páginas completas **sin HTML escrito a mano**, solo mediante nodos JS (objetivo final del módulo según Sara).
- 

### 4 Formas de trabajar con JS



## 4.1 Dentro del documento HTML

En la sección `<head>` o `<body>` :

```
<head>
  <script>
    // instrucciones JS
  </script>
</head>
```

## 4.2 Archivo externo `.js` (la forma recomendada)

```
<script src="datos.js"></script>
```

Sara recomienda trabajar **siempre con archivos externos** para mantener:

- limpieza
- orden
- separación HTML / CSS / JS

---

## 5 Salida de datos

### 5.1 Usando el objeto `window`

Crea una ventana emergente:

```
window.alert("Hola");
```

### 5.2 Consola del navegador

(No interrumpe la ejecución, ideal para depuración):

```
console.log("Hola");
```

### 5.3 Escribir en el documento

No recomendable en producción:

```
document.write("Hola");
```

---

## 6 Pedir datos al usuario

Antes de ver formularios, Sara explica cómo solicitar valores con `prompt` :

```
let valor = window.prompt("Introduce valor:");
```

Todo lo que devuelve `prompt` es **string**.

(La conversión numérica llegará más adelante: `parseInt` , `parseFloat` ).

---

## 7 Variables en JavaScript

## 7.1 var (evitar)

- Declaración antigua.
- Variables globales por defecto.
- Peor gestión de memoria.
- No respeta el *block scope*.

## 7.2 let (buena práctica)

- Respeto el ámbito de bloque `{}` .
- Mayor control de memoria.
- Sara insiste:  
en 2025 casi todo se escribe con `let` o `const` , nunca con `var` .

## 7.3 Ejemplo:

```
let numero1;  
numero1 = 23;  
numero1 = "hola";
```

Recordatorio: JS permite cambiar el tipo dinámicamente.

---

## 8 Operadores

### 8.1 Aritméticos

`+` `-` `*` `/` `%`

### 8.2 Comparación

`>` `<` `>=` `<=` `!=` `==` `===`

(Sara todavía no ha diferenciado `==` vs `===`, eso llegará más adelante en sintaxis estricta.)

### 8.3 Lógicos

`&&` (and)

`||` (or)

`!` (not)

### 8.4 Asignación

`=` `+=` `-=` `*=` `/=` `%=`

---

## 9 Estructuras condicionales

### 9.1 Estructura básica `if` / `else` :

```
if (condicion) {  
    // instrucciones  
}  
else {  
    // instrucciones alternativas  
}
```

## 9.2 Estructuras condicionales compuestas

Se amplían con:

```
else if (condicion) {  
  
}
```

Sara muestra ejemplos comparando números mediante `window.prompt` :

```
let numero = window.prompt("Introduce número");  
let numero4 = window.prompt("Introduce otro número");  
  
if (numero > numero4) {  
    window.alert("El número mayor es numero");  
}  
else if (numero == numero4) {  
    window.alert("Los números son iguales");  
}  
else {  
    window.alert("El número no es mayor");  
}
```

---

## 10 Ejecución del código en VS Code

Sara explica cómo abrir el archivo `.html` en el navegador para ver:

- alertas
- resultados de consola
- errores de sintaxis
- comportamiento de scripts externos

Código mostrado en clase:

```
<!doctype html>  
<head>  
</head>  
    <script>  
        // ejemplo ventana alert  
        window.alert("Hola mundo");  
    </script>  
    // link a script js  
    <script src="javascript1.js"></script>  
<body>  
</body>  
</html>
```

---

## 1 1 Buenas prácticas

- Utilizar `let` y `const` ; evitar `var` .

Porque `var` crea variables globales de forma implícita, ignora el ámbito de bloque y puede provocar

errores difíciles de depurar.

`let` respeta los bloques `{}` y `const` garantiza que el valor no cambiará.

- **Separar HTML / CSS / JS en archivos independientes.**

Esto mejora la organización, reduce errores, facilita la lectura del proyecto y permite que el navegador optimice la carga.

Un proyecto limpio es más fácil de escalar cuando lleguemos al DOM.

- **Evitar `document.write()` salvo en demostraciones simples.**

Sobrescribe el documento si se usa después de la carga, rompe el DOM y no se utiliza en desarrollos profesionales.

- **Validar valores obtenidos mediante `prompt()`.**

`prompt` siempre devuelve texto y puede devolver `null`.

Hay que convertir y comprobar los datos antes de operar con ellos (`parseInt`, `parseFloat`, comprobación de vacío, etc.)

- **Mantener ordenados los archivos JS antes de pasar al DOM.**

Tener claro dónde está cada funcionalidad evita confusiones al empezar con la jerarquía de nodos, eventos, selectores y manipulación del árbol DOM.

---

## Próxima clase (Clase 4)

Sara adelanta que:

- Se finalizará la sintaxis básica de JS.
  - Empezaréis a trabajar con **el DOM**:
    - nodos
    - jerarquías
    - acceso a elementos
    - creación dinámica de contenido
  - El objetivo final es **crear una página completa manipulando nodos JS**, sin escribir HTML directamente.
-